AFRL-RH-FS-TR-2014-0044

Modeling Laser Damage Thresholds Using the Thompson-Gerstman Model

Chad A. Oian
Austin Kane
TASC, Inc.

Paul Kennedy
Robert J. Thomas
711th Human Performance Wing
Human Effectiveness Directorate
Bioeffects Division
Optical Radiation Bioeffects Branch

October 2014

Interim Report for December 2013 to December 2014

DESTRUCTION NOTICE – Destroy by any method that will prevent disclosure of contents or reconstruction of this document.

Air Force Research Laboratory
711th Human Performance Wing
Human Effectiveness Directorate
Bioeffects Division
Optical Radiation Bioeffects
Fort Sam Houston, Texas 78234

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

"Modeling Laser Damage Thresholds Using the Thompson-Gerstman Model"

(AFRL-RH-FS-TR- 2014 - 0044) has been reviewed and is approved for publication in accordance with assigned distribution statement.

ROCKWELL.BENJAMIN.A.1231305358 Digitally signed by ROCKWELL.BENJAMIN.A.1231305358 DN: c=US, o=U.S. Government, ou=DoD, ou=PKI, ou=USAF, cn=ROCKWELL.BENJAMIN.A.1231305358 Date: 2014.10.22 15:30:57 -05'00'

Benjamin A. Rockwell
Work Unit Manager
Optical Radiation Bioeffects Branch

POLHAMUS.GARRETT.D.1175839484 Digitally signed by POLHAMUS.GARRETT.D.1175839484 DN: c=US, o=U.S. Government, ou=DoD, ou=PKI, ou=USAF, cn=POLHAMUS.GARRETT.D.1175839484 Date: 2015.01.20 13:52:34 -06'00'

GARRETT D. POLHAMUS, DR-IV, DAF
Chief, Bioeffects Division
Human Effectiveness Directorate
711th Human Performance Wing
Air Force Research Laboratory

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
20/10/2014	Interim Technical Report	December 2013-December 2014

4. TITLE AND SUBTITLE
Modeling Laser Damage Thresholds Using the Thompson-Gerstman Model

5a. CONTRACT NUMBER
FA8650-14-D-6519

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
0602202F

6. AUTHOR(S)
Chad A. Oian, Austin Kane, Paul Kennedy, Robert J. Thomas

5d. PROJECT NUMBER
7757

5e. TASK NUMBER
HD

5f. WORK UNIT NUMBER
04 H0BC

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
711th Human Performance Wing
Human Effectiveness Directorate
Bioeffects Division
Optical Radiation Bioeffects
Fort Sam Houston, Texas 78234

TASC, Inc.
4141 Petroleum Rd
Fort Sam Houston, Texas 78234

8. PERFORMING ORGANIZATION REPORT

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory
711th Human Performance Wing
Human Effectiveness Directorate
Bioeffects Division
Optical Radiation Bioeffects
Fort Sam Houston, Texas 78234

10. SPONSOR/MONITOR'S ACRONYM(S)
711 HPW/RHDO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-RH-FS-TR-2014-0044

12. DISTRIBUTION / AVAILABILITY STATEMENT
Pending Distribution A: Approved for public release; distribution unlimited. PA Case No: TSRL-PA-2014-0013.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
As lasers become more pervasive, the dangers posed by laser radiation to the human retina have increased. Computer aided modeling of laser tissue interaction in the retina allows for researchers to simulate parameter ranges where exposure can cause damage. While this approach has been studied using a variety of methods, one of the most referenced models has been the Thompson-Gerstman melanin granule model. Due to limited computing resources at the time, the original FORTRAN version of this model was implemented as a stand-alone serial code and was only able to model single-pulse exposures. A new C++ version of the Thompson-Gerstman model has been implemented, which expands both the functionality and portability of the method. The source type has been expanded to include multi-pulse as well as single pulse. Additionally, the functions developed by Thompson and Gerstman are now part of a library that can be accessed through other modeling software.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
Unclassified					James Beilby
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**		42	**19b. TELEPHONE NUMBER** *(include area code)*
U	U	U			NA

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. 239.18

This Page Intentionally Left Blank

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

As lasers become more pervasive, the dangers posed by laser radiation to the human retina have increased. Computer aided modeling of laser tissue interaction for energies deposed in the retina allows for researchers to simulate parameter ranges where exposure can cause damage. While this approach has been studied using a variety of methods, one of the most referenced models has been the Thompson-Gerstman melanin granule model. Due to limited computing resources at the time, the original FORTRAN version of this model was implemented as a stand-alone serial code and was only able to model single-pulse exposures. A new C++ version of the Thompson-Gerstman model has been implemented, which expands both the functionality and portability of the method. The source type has been expanded to include multi-pulse as well as single pulse. Additionally, the functions developed by Thompson and Gerstman are now part of a library that can be accessed through other modeling software.

1 INTRODUCTION

Laser radiation in the visible and near infrared (400-1400 nm) region of the spectrum poses a unique risk to the human eye. A collimated laser beam incident on the cornea will be focused to a small spot on the retina with a transmission between the cornea and retina of approximately 90% at 550 nm [1]. As light passes through the retina, the layers of tissue will absorb energy, which could lead to damage. Damage to the retina most likely occurs in the retinal pigmented epithelium (RPE) layer where 50% of 600 nm light is absorbed [2]. RPE cells in the pigment layer, located at the back of the retina, may be irreparably damaged if laser radiation deposition is sufficiently intense. Damage to cells in the RPE layer may result in loss of vision sensing in the affected region of the retina.

Damage can be caused by three distinct mechanisms: photothermal, photomechanical, and photochemical. Photothermal damage occurs when the temperature of tissue rises to levels where protein denaturation or enzyme inactivation occurs. This type of damage is produced with a long-pulse duration ($t > 10$ μs) where substantial energy is deposited slowly. Photomechanical damage occurs under short and ultrashort pulses (10 ρs $< t < 10$ μs) where energy is deposited quickly, causing mechanical stress or micro-cavitation within the tissue and cells.

An accurate computational model of the laser-tissue interaction, which accounts for energy absorption and thermal diffusion, could be used to predict damage thresholds based on the properties of the laser source. The relationship between beam characteristics such as wavelength, pulse duration, beam profile, and spot size could then be compared to damage thresholds and lesion size, which would give insight into exposure thresholds in regimes where no laser damage studies have been conducted. This tool would have the added benefit of reducing the dependence on animal studies for determining damage thresholds.

The topic of modeling photothermal injury to the retina has been evaluated by a wide range of authors [3-8]. While several possible simulations have emerged to model the laser-tissue interaction in the retina, a “granular model” has advantages over other models, which try to numerically solve the heat equation assuming homogeneous retinal layers and homogeneous heat absorption. At visible and near-infrared wavelengths most optical absorption in the RPE is within the intracellular melanin-impregnated granules known as melanosomes. For pulses shorter than 0.1 ms, the assumption of homogeneous temperature within layers breaks down, as heat near the energy absorbing melanosomes is much greater than in surrounding tissue.

A number of researchers have developed granular models [9-13] in order to address the issue of non-homogeneous heating of the RPE. The most advanced of the granular models, and the one most often referenced by other researchers since its development, is the Thompson-Gerstman melanin granule model [14]. Instead of solving for temperature across the domain through finite

element or finite difference meshing, the Thompson-Gerstman model instead solves the analytic heat profile of a single melanosome, then calculates the temperature at any given point in the medium through superposition of thermal energy. This report documents the creation of a C++ implementation of the Thompson-Gerstman model by Austin Kane, which replaces, and improves on, the original FORTRAN implementation created in 1996 by Dr. Randy Thompson. The C++ implementation should allow for greater utility and easier integration into other laser-tissue damage models.

2 MODELING LASER-TISSUE INTERACTION

2.1 Thompson-Gerstman Model

An analytic expression for heated spheres in a homogeneous infinite medium, originally derived by Goldenburg in the 1950's [9,10], forms the basis of the Thompson-Gerstman granular model. A granular model can also be modified to include photomechanical damage from bubble formation near the granules, which has been presented in other models [11]. However, the Thompson-Gerstman model considers only photothermal effects. While many granule models assume melanosomes of zero diameter to reduce the complexity of the analytical expression, this assumption breaks down for ultra-fast pulses where calculating the temperature rise near granules becomes inaccurate [12]. The Thompson-Gerstman model has melanosomes of finite size, where the analytical solution of the heat equation has been derived at the granule center, at radial distances from the center that are inside the granule, at the granule surface and at radial distances that are outside the granule.

The model developed by Thompson and Gerstman [13,14] makes several simplifying assumptions about material properties and about the nature of tissue-laser interaction in the retina. First, it assumes melanosomes are homogenous spheres randomly distributed in an infinite volume of liquid water. Second, it assumes absorption of laser energy takes place only within melanin granules and that this energy absorption is homogeneous. Third, the melanin granules also have the thermal properties of water. This assumption can be removed in favor of a more accurate solution [9], but then thermal superposition would no longer be valid.

The governing equation for heat transfer from a single granule with spherical symmetry is shown in Equation (2.1).

$$\frac{1}{r}\frac{\partial^2}{\partial r^2}(rT) + \frac{A_o(r,t)}{k} = \frac{1}{\alpha}\frac{\partial T}{\partial t} \tag{2.1}$$

The heat equation is expressed in terms of **r,** the scalar distance from the granule center [cm], **T** is temperature [◦C], **t** is time [sec], **α** is the thermal diffusivity of the medium [cm^2/sec], **k** is the thermal conductivity [J/cm ◦C sec], and $\mathbf{A_o}$ is a source term defined in Equations (2.2) and (2.3).

$$\mathbf{A_o} = \frac{\mathbf{3I_o}}{\mathbf{4a\tau_p}}\left[\mathbf{1} - \frac{\mathbf{1}}{\mathbf{2\alpha_m^2 a^2}}\left(\mathbf{1} - \mathbf{exp}(\mathbf{-2a\alpha_m})\,(\mathbf{1} + \mathbf{2\alpha_m a})\right)\right] \tag{2.2}$$

$$\mathbf{A_o(r,t)} = \begin{cases} \mathbf{A_o} \;\; \mathbf{if\ 0 \geq r \geq a; 0 \geq t \geq \tau_p} \\ \mathbf{0} \;\; \mathbf{otherwise} \end{cases} \tag{2.3}$$

The variable $\mathbf{I_o}$ is the average retinal fluence [J/cm^2], **a** is the granular radius [μm], $\boldsymbol{\tau_p}$ is the pulse length [sec], and $\boldsymbol{\alpha_m}$ is the wavelength-dependent melanosome absorption coefficient [cm^{-1}].

2.2 FORTRAN Code

The FORTRAN implementation of the Thompson-Gerstman model provided computational analysis of damage given a set of conditions and parameters that could be approximated from data gathered experimentally. A rectangular coordinate system was used to define the locations of melanin granules and to reference the spatial points at which temperature rises were computed. The origin of the coordinate system is defined as the center of the rectangular volume shown in Figure 1.

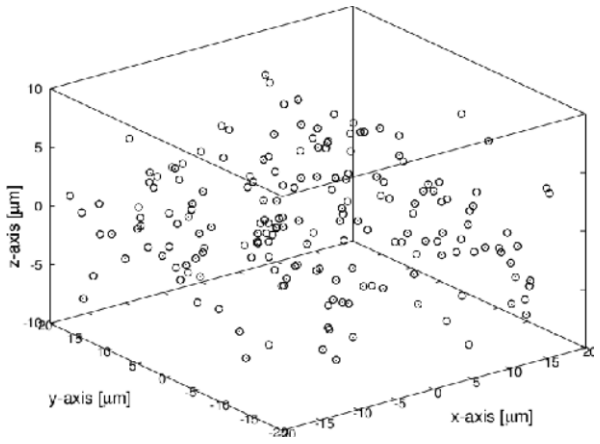
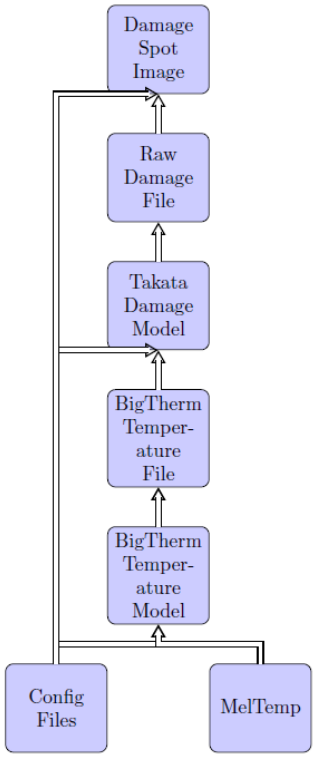


Figure 1: A Profile of the Distribution of Melanin Granules About the RPE Layer

The structure of the FORTRAN code can be described in three steps:



1) Input Parameters: The initial conditions needed for the temperature calculations, as well as a set of example values, are shown in Table 1. Values related to simulated domain and physical parameters are used to calculate other variables needed for initialization. These parameters are stored in "Config Files" shown in Figure 2.

2) Thermal Solution: A temperature value is calculated for each grid point the user specifies over an interval of time such that gradients in thermal diffusion have significantly decreased. The time-temperature values are obtained by a superposition of the thermal energy contributed by each granule. The position, time, and temperature data is then saved in a text file shown in Figure 2.

3) Damage: Since the aim of the model is to determine if damage has occurred under laser radiation exposure, time-temperature data alone are insufficient. The Arrhenius damage integral was selected for its usefulness in modeling thermal damage to biological tissue. The damage data are saved as a "Raw Damage File" shown in Figure 2.

Figure 2: Flow Diagram of the FORTRAN Thompson-Gerstman Model

Table 1: Initial Values Needed in FORTRAN Version of Thompson-Gerstman Model

Description	Value
X domain	[-20,20] μm
Y domain	[-20,20] μm
Z domain	[-7,7] μm
Number of melanin granules	186
Simulation time	[0,2] ms
Corneal fluence	3×10^{-5} J/cm^2
Pulse duration	3×10^{-6} s
Thermal diffusivity of the medium	0.00133 cm^2/s
Thermal conductivity of the medium	0.00556 J/cm-C-sec
Beam diameter at cornea	0.7 cm

2.3 Source Types

The type of source exposure can also be designated as one of three options: gaussian, top-hat, or annular. The gaussian profile is most commonly used as the assumed profile of most laser exposures and represents an ideal beam. A top-hat profile has a single region of high intensity surrounded by a region of zero intensity without the smooth transition seen in the gaussian profile. An annular beam resembles a ring or donut-shaped beam similar to the top-hat profile with a small region of zero intensity in the center of the beam profile.

Examples of thermal results using the parameters in Table 1 are shown in Figure 3. The false color images show temperature rise in an x-y slice at the beam focus.

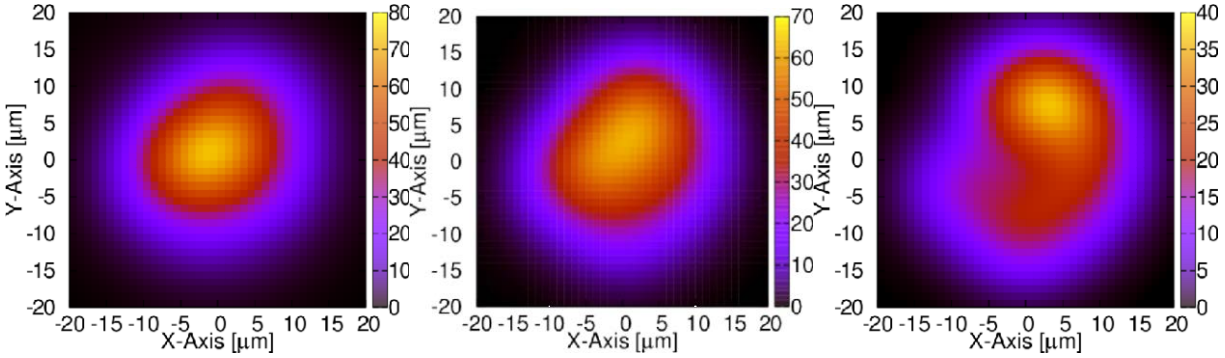


Figure 3: Thermal Profiles Resulting from (a) Gaussian Beam, (b) Top-hat Beam and (c) Annular Beam

2.4 Damage

While it is true that tissue heated to the vaporization point of water will likely cause damage, it is also possible for damage to occur at lower temperatures. Experiments indicate that temperature elevations of just 10° C can cause cellular damage if maintained for a long enough time. The Arrhenius damage integral approach [15] assumes damage occurs by an active rate process described by

$$\Omega(r) = A \int_{t_i}^{t_f} \exp\left(-\frac{E}{RT(r,t)}\right) dt. \qquad (2.4)$$

The variable **Ω** represents the damage level, **E** is the activation energy [energy per mole], **R** is the gas constant, **T** is the temperature, and $\mathbf{t_i}$ through $\mathbf{t_f}$ represents the lower and upper limit of time evaluated. The constants

$$A = 1.3\ x\ 10^{99} \quad \left[\frac{1}{s}\right]$$

and

$$E = 1.5\ x\ 10^{5} \quad \left[\frac{cal}{mol}\right]$$

were found [16] to give a simple value for $\boldsymbol{\Omega}$ to determine if damage had occurred.

$$\begin{cases} damage & \Omega \geq 1 \\ no\ damage & \Omega < 1 \end{cases}$$

Using the time-temperature data of the exposure simulation, the Arrhenius integral is solved numerically by Simpson's rule. An example of a damage spot calculated using this method is shown in Figure 4. The damage spot was calculated by the Arrhenius integral for exposure to a gaussian beam with input parameters given in Table 1.

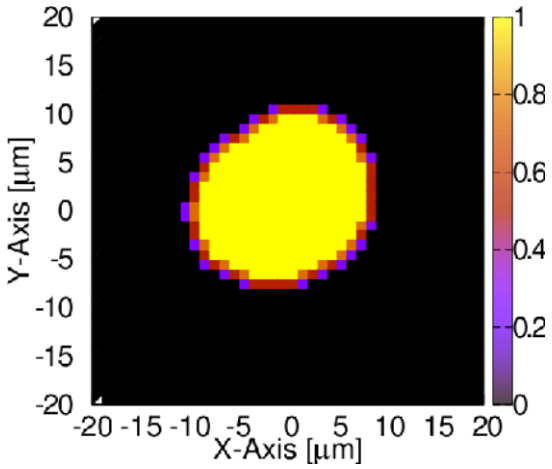


Figure 4: Damage Spot from Gaussian Beam

3 IMPROVEMENT OF THOMPSON-GERSTMAN MODEL

3.1 C++ Implementation

Several improvements were suggested to the original FORTRAN implementation of the Thompson-Gerstman model. First, the method needed to be converted to a language which allowed for added usability and easy integration into other programs. Second, the temperature data should be passed directly to the damage integral instead of reading and writing time-temperature data to file between each step. Third, the user should be able to provide a range of values as input parameters instead of requiring modifying the input file for each run. Finally, an option of multiple-pulse exposures could be added to expand the application of this model beyond the single-pulse realm. With these criteria, the C++ language was selected to build the new implementation of the model.

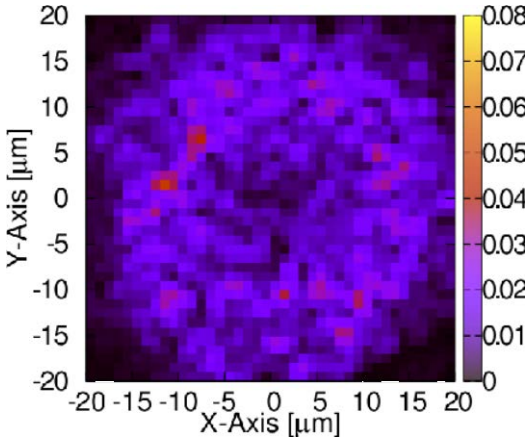
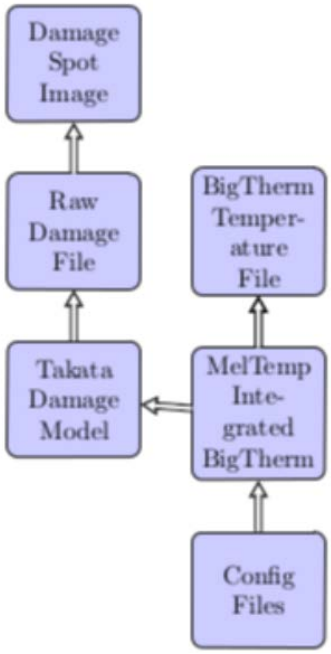


Figure 5: Absolute Temperature Difference

A validation of the C++ implementation was performed by direct comparison to the thermal results given by the FORTRAN code. This thermal results comparison, shown in Figure 5, produced maximum differences in temperature around 0.08 °C. This gives a maximum relative percent error of 0.04%. This difference is likely due to minor differences between how the two languages handle float point precision.

3.2 C++ Code Structure



- The C++ version, through use of modular design, was able to significantly streamline the process of generating damage files. Many of these changes reflected the intended usage and computing power available for the new implementation.

- This version of the Thompson-Gerstman model was intended to be a modular tool fit for integration into other computational models. This adds usability to the standalone code which can now be ported as a small function library. The new top-down approach of the structure is shown in Figure 6.

Figure 6: Top Down Overview of C++ Version of Thompson-Gerstman Model

3.3 Variable Input Parameters

The new implementation has the ability to iterate the thermal solver algorithm over any number of different parameters such as corneal fluence, laser power, number of points, and other input parameters of significance. This modularized code might be used to optimize input parameters in the model to produce effects which match experimental values. Figure 7 shows the change in temperature in the medium at the beam focus shortly after a single pulse. A range of corneal fluence values between 3.0 – 4.5 μJ/cm^2 was provided while holding other variables constant. As expected, a linear relationship between corneal fluence and temperature exists directly after the pulse.

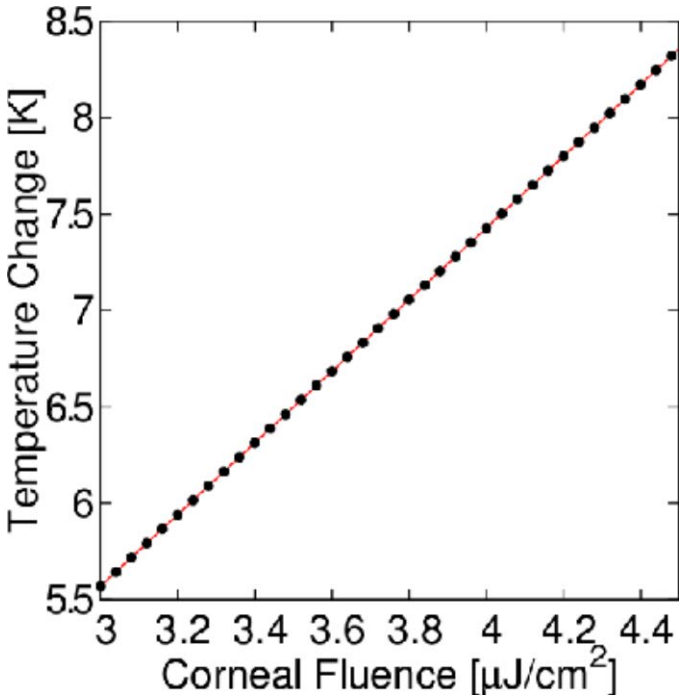


Figure 7: Temperature Rise at Beam Focus

3.4 Multi-Pulse Capability

While meaningful analysis can be performed by varying parameters for a single-pulse exposure, multiple-pulse trains present several additional parameters to compare damage thresholds. Due to the unique way the Thompson-Gerstman model handles temperature changes from a single pulse, implementing a multi-pulse function simply requires additional iterations of the superposition temperature solver. This allows for variance in number of pulses as well as for the time between pulses.

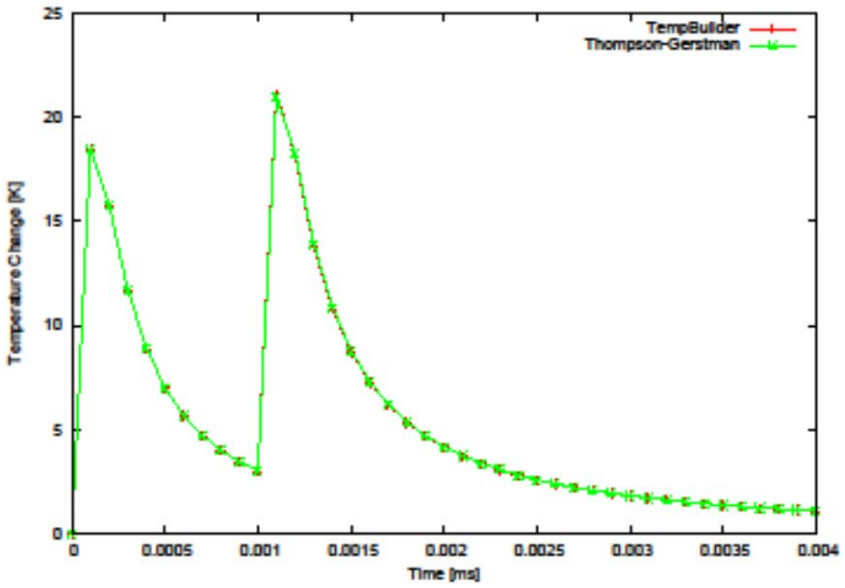


Figure 8: Multi-Pulse Validation

Validation of this method was performed by direct comparison to another multiple-pulse computational solver, which has been independently verified. Figure 8 shows the direct comparison of the multi-pulse Thompson-Gerstman algorithm (green) next to the verified TempBuilder (red) thermal profile. These results show good agreement, which supports the validity of the new C++ Thompson-Gerstman code for multiple-pulse exposures.

4 DISCUSSION

With the growing number of laser systems being used by both military and civilians, laser radiation damage to the retina is a growing concern. Damage to tissue and loss of vision can occur under a variety of beam conditions, wavelengths, and exposure times. Developing a set of laser safety standards requires testing exposure conditions to define safe limits of laser use. Simulated modeling of the parameter ranges where exposure conditions cause photothermal damage will better help researchers predict damage.

The Thompson-Gerstman model is able to give an estimate of whether damage has occurred by making several assumptions that are not physically valid, but may provide model results that are a reasonable match to actual biological damage data. First, the model is not given a thermal boundary condition, so the exact rate of thermal energy transmitted to surrounding tissue is neglected. For exposures greater than 1 second, or long simulation times where the diffusion term dominates the heat equation, this assumption would likely cause the model's temperature to be lower than expected. Additionally, since photomechanical and photochemical damage are not included in the model, short pulses and temperatures at or near vaporization should be ignored as non-valid. However, since photothermal damage covers a wide range of possible conditions,

other damage mechanisms are not needed as long as the model is used in a valid laser pulse range.

The positive aspects of this model lie mostly in its numerical simplicity and speed of simulation. Since computing temperature is simplified using superposition of analytic solutions, long lists of simultaneous equations or numerical methods of solving partial differential equations are unnecessary. As a result, the computational effort is reduced to several loops of simple arithmetic. This simplifies what could be an exhaustive parameter study into a series of non-intensive computational steps.

5 CONCLUSIONS

The work which began with the FORTRAN implementation of the Thompson-Gerstman model has been expanded to a C++ modular model that possesses a function library capable of interfacing with other laser-tissue damage models. This modular model has improved the usability of the method for conducting damage threshold research. The granular model described here and detailed in their paper [14] has proven to be a fast and efficient method at predicting retinal damage using a wide range of input parameters. Additional developments of multi-pulse functionality and solving exposures over large ranges of input parameters greatly increase the application and research potential of this method. Direct comparison validation has been performed on the results of the FORTRAN and C++ codes, here showing good agreement between the two computational codes. Results from this model could be compared to known biological results, which would possess insight into thresholds in regimes where no laser damage studies have been conducted. This tool would have the added benefit of reducing the complexity and sensitivity of attaining results when depending on animal studies for determining damage thresholds.

6 REFERENCES

1. Geeraets, W.J. and R. E. Berry. 1968. Ocular spectral characteristics as related to hazards from lasers and other light sources. *Am. J. Ophthalmol*. **66**, 15-20.
2. Boettner, E. A. and J. R. Wolter. 1962. Transmission of the ocular media. U.S. Air Force Technical Documentary Report MRL-TDR-62-34.
3. Birngruber, R. 1980. Thermal modeling in biological tissues. In *Lasers in Biology and Medicine*, F. Hillenkamp, R. Patresi and C. A. Sacchi (Eds.). *NATO Advanced Study Institute, Series A – Life Sciences*, Vol. **34**, pp. 77-97. New York: Plenum Press.
4. Birngruber, R., V.-P. Gabel and F. Hillenkamp. 1983. Experimental studies of laser thermal retinal injury. *Health Phys*. **44**, 519-531.
5. Birngruber, R., F. Hillenkamp and V.-P. Gabel. 1985. Theoretical investigations of laser thermal retinal injury. *Health Phys*. **48**, 781-796.
6. Allen, R. G. 1979. Retinal thermal injury. Non-ionizing Radiation – Proceedings of an ACGIH Topical Symposium, pp. 161-168.
7. Welch, A. J. 1984. The thermal response of laser irradiated tissue. *IEEE J. Quantum Electron*. **QE-20**, 1471-1481.
8. McKenzie, A. L. 1990. Physics of thermal processes in laser-tissue interaction. *Phys. Med Biol*. **35**, 1175-1209.
9. Goldenberg, H. 1951. A problem in radial heat flow. *British J. Appl. Phys*. **2**, 233.
10. Goldenberg, H. and C. J. Tranter. 1952. Heat flow in an infinite medium heated by a sphere. *British J. Appl. Phys*. **3**, 296.
11. Hansen, W. P. and S. Fine. 1968. Melanin granule models for pulsed laser induced retinal injury. *J. Appl. Opt*. **7**, 155-159.
12. Zheltov, G., V. Glazkov and A. Podol'tzef. 1989. Retinal damage from intense visible light. Health Phys. **56**, 625-630.
13. Thompson, C. R. 1994. Melanin granule model for heating of tissue by laser. Laser-Tissue Interaction V. *Proc*. SPIE **2134A**, 66-77.
14. Thompson, C. R., Gerstman, B. S., Jacques, S. L., and Rogers, M. E. 1996. Melanin Granule Model for Laser-Induced Thermal Damage in the Retina. *Bulletin of Mathematical Biology*. Vol. **58**, No. 3, 513-553.
15. Henriques, F. C. 1947. Studies of thermal injury. *Arch. Path*. **43**, 489-502.
16. Welch, A. J. and G. D. Polhamus. 1984. Measurement and prediction of thermal injury in the retina of the rhesus monkey. *IEEE Trans. Biomed. Eng*. **BME-31**, 1471-1481.

7 APPENDIX: THOMPSON GERSTMAN MODEL C++

7.1 Top Level Function

```
#include<math.h>
#include<ctime>
#include <boost/test/unit_test.hpp>
#include<iostream>
#include<fstream>
#include<vector>
#include<string>
using namespace std;
#include "TGlib.cpp"
#include <boost/property_tree/ptree.hpp>
#include <boost/property_tree/ini_parser.hpp>
#include<omp.h>

int main(int argc, char* arg[]){
      BigTherm Therm(arg[1]);
      Therm.TakataDamage();
      return 0;
}
```

7.2 Header File

```
#include<vector>
#include <boost/property_tree/ptree.hpp>
#include<string>
using namespace std;

class granule {
public:

//---------- These parameters are given in .inp1 files------------
      double a; //Granular radius

//---------- End parameters given by .inp1 files-------------

//---------- These parameters are givin in .inp2 files -----
      double pulsedur; //pulse duration in seconds
      double alpha; // Thermal Diffusivity of medium in cm**2/sec
      double cond; // Thermal conductivity of medium in j/cm-Csec

//---------- End Parameters from .inp2 files --------------------
//used in (orig,in,surf,out,vanilla)temp functions, calculated in
//bigthrem from other input parameters

      double A0;
      double OutTemp(double r, double t);
      double SurfTemp(double t);
      double InTemp(double r, double t);
      double OrigTemp(double t);
      double Temp(double r, double t);
      double MelTemp(double r, double t);
      double tmin, PulseSep;
      int PulseNum;
      vector<double> PulseVec;
};

class BigTherm {
      public:
            BigTherm(char* configfilename);
            vector<int> elapsed, tarray, guess;
            int k, s, improf, n,TempFlag, SeedFlag;
            int xnum, ynum, znum, tnum, melnum, rnum;
            vector<int> test;
            double BoxLength, BoxDepth, MelDens;
            double xmin, xmax, xsize;
            double ymin, ymax, ysize;
            double zmin, zmax, zsize;
            double tmin, tmax, tsize;
            vector< vector< double > > Pos;
            vector< double >  weight, PulseVec;
            vector< vector< double > > VecTemp;
            vector< double > rsize;
            double  dx;
            double ran1, ran2, ran3;
            double irr,Tbody, absorb;
            double Temp, x, y, z, t, r;
```

```
            double sigma, corspot, Focus;
            double spotsize,MelTemp,trans,dobs;
            double Qtot, Qlost, Qkept;
            granule granny;
            boost::property_tree::ptree configfile;
            string MelPlacementFile;
            string TempOutputFile;
            string DamageOutputFile;
            double BigThermTempAt(double x, double y, double z, double t);
            // this function is to build the function again if you change any
            // variables that affect weight or vectemp
            void ReConstruct(void);
            void BigTherm2(void);
            void BigTherm2(double outputxmin, double outputxmax,int outxnum,
double outputymin, double outputymax, int outynum, double outputzmin, double
outputzmax, int outznum);
            int PulseNum;
            double PulseSep;
            void TakataDamage(void);
            int TakataDamageAt(double x, double y, double z);
            int DamageInfo;
            int numthreads;
};

void takatadamage(char* configfilename);
```

7.3 Library File

```
#include<math.h>
#include"TGlib.h"
#define _USE_MATH_DEFINES
#include<limits>
#include<vector>
#include<fstream>
#include <boost/property_tree/ptree.hpp>
#include <boost/property_tree/ini_parser.hpp>
#include<string>
#include<iostream>
#include"omp.h"
#include <time.h>
using namespace std;
double Gmma(double aa, double bb, double cc){
if(bb == 0.0) {
//-------------gigamma function --------------------------
if(cc > 30.0)
      cc = 30.0;
//-------------gammaser function -------------------
//-----modified(un-normalized) version of GSER from numerical recipes ----
/*
Modification of GSER is ignoring (removing) the "gln" variable, and the
function that calculates it gammln(a). this is used in the sum evaluating the
exponential for del < sum *EPS, as subtraction by the natural log of the
complete gamma function inside the exponential acts the same as dividing by
the same value.
*/
      const int ITMAX = 100;
      const double EPS = numeric_limits<double>::epsilon();
      double sum, del,ap;
      if(cc <= 0.0){
            if(cc<0.0){
                  return -1;
                  //fail in a mysterious way
            }
            else
                  return 0.0;
      }
      else{
            ap = aa;
            del = sum = 1.0/aa;
            for(int n = 0; n <ITMAX; n++){
                  ap += 1.0;
                  del *= cc/ap;
                  sum += del;
                  if(fabs(del) < fabs(sum)*EPS){
                        return sum*exp(-cc+aa*log(cc));
                  }
            }
      }
}
}

double LagL(double a, double b, double c){
```

```
double d, gm1, gm2, Gamma, answer;
d = -c;
gm1 = -0.5;
gm2 = 0.0;
Gamma = Gmma(gm1,gm2,d);
answer = - (Gamma * sqrt(d) * (1.0 - 2*c/3) + (2*exp(c))/3)/M_PI;
return answer;
}

//calculates the temperature outside the granule
double granule::OutTemp(double r, double t){
double Outterm[4], gm[2], lg[2];
double OutTheta;
gm[0] = 0.5;
gm[1] = 0.0;
lg[0] = 1.5;
lg[1] = -0.5;
double gamma = Gmma(gm[0],gm[1],(((-a + r)*(-a + r))/(4.0*alpha*t)));
      Outterm[0]    =    ((a*t*((sqrt(M_PI)*((-a    +    r)*(-a    +    r)    +
2.0*alpha*t))/(2.0*alpha*t)   -   (2.0*(-a   +   r)*(0.5*exp((-((-a   +   r)*(-a
+r))/(4.0*alpha*t)))  +  (sqrt(((-a  +  r)*(-a  +r))/(alpha*t))*(((-a  +  r)*(-a
+r))       +       2.0*alpha*t)*       gamma)/(4.0*((-a       +       r)*(-
a+r)))))/(sqrt(alpha)*sqrt(t))))/(sqrt(M_PI)));
      Outterm[1] = (a*a*a - 3.0*a*a*r + 3.0*a*r*r - (r*r*r) + 6.0*a*alpha*t -
6.0*alpha*r*t                                                                 +
6.0*(alpha*sqrt(alpha))*sqrt(M_PI)*(t*sqrt(t))*LagL(lg[0],lg[1],-((-a  +  r)*(-
a + r))/(4.0*alpha*t)))/(6.0*alpha);
      Outterm[2] = (-a*a*a - 3.0*a*a*r - 3.0*a*r*r - r*r*r - 6.0*a*alpha*t -
6.0*alpha*r*t                                                                 +
6.0*(alpha*sqrt(alpha))*sqrt(M_PI)*(t*sqrt(t))*LagL(lg[0],lg[1],(-(a  +  r)*(a
+r))/(4.0*alpha*t)))/(6.0*alpha);
      Outterm[3]    =    (a*t*((sqrt(M_PI)    *    ((a    +    r)*(a    +    r)    +
2.0*alpha*t))/(2.0*alpha*t)  -  (2.0*(a  +  r)*  (0.5*exp(-((a  +  r)*(a  +
r))/(4.0*alpha*t)) + (sqrt(((a + r)*(a + r))/(alpha*t))*((a + r)*(a + r) +
2.0*alpha*t)*Gmma(gm[0], gm[1], ((a + r)*(a + r))/(4.0*alpha*t)))/(4.0*(a +
r)*(a + r))))/(sqrt(alpha)*sqrt(t))))/(sqrt(M_PI));
OutTheta    =    alpha*A0*(Outterm[0]    -    Outterm[1]    +    Outterm[2]    +
Outterm[3])/(2.0*cond);
return OutTheta/r;
}

// calculates temperature on the surface of the granual
double granule::SurfTemp(double t){
double Surffnc[4];
double SurfTheta, gm[2], lg[2];
gm[0] = 0.5;
gm[1] = 0.0;
lg[0] = 1.5;
lg[1] = -0.5;
Surffnc[0] = t;
Surffnc[1] = t*(sqrt(M_PI)*(2.0*a*a + alpha*t)/(alpha*t) - 4.0*a*(0.5*exp(-
(a*a)/(alpha*t))          +          sqrt((a*a)/(alpha*t))*(2.0*a*a          +
alpha*t)*Gmma(gm[0],gm[1],(a*a)/(alpha*t))/(4.0*a*a))/(sqrt(alpha)*sqrt(t)))/
(sqrt(M_PI));

Surffnc[2] = 4.0*(t*sqrt(t))/(3.0*sqrt(M_PI));
```

```
Surffnc[3] = -4.0*(a*a*a)/(3.0*alpha*sqrt(alpha)) - 2.0*a*t/sqrt(alpha) +
sqrt(M_PI)*t*sqrt(t)*LagL(lg[0],lg[1],-((a*a)/(alpha*t)));
Surffnc[4] = a*alpha*A0*(Surffnc[0] + Surffnc[1])/(2.0*cond) +
alpha*sqrt(alpha)*A0*(Surffnc[3] - Surffnc[2])/(2.0*cond);
return Surffnc[4]/a;
}

// Calculates temperature inside the granual
double granule::InTemp(double r, double t){
double Interm[5], gm[2], lg[2];
double Intheta;
gm[0] = 0.5;
gm[1] = 0.0;
lg[0] = 1.5;
lg[1] = -0.5;
Interm[0] = r*t;
Interm[1] = (-a*a*a + 3.0*a*a*r - 3.0*a*r*r + r*r*r - 6.0*a*alpha*t +
6.0*alpha*r*t + 6.0*alpha*sqrt(alpha)*sqrt(M_PI)*t*sqrt(t)*LagL(lg[0],lg[1],-
((a - r)*(a - r))/(4.0*alpha*t)))/(12.0*alpha);
Interm[2] = (-a*a*a - 3.0*a*a*r - 3.0*a*r*r - r*r*r - 6.0*a*alpha*t -
6.0*alpha*r*t + 6.0*alpha*sqrt(alpha)*sqrt(M_PI)*t*sqrt(t)*LagL(lg[0],lg[1],-
((a + r)*(a + r))/(4.0*alpha*t)))/(12.0*alpha);
Interm[3] = a*t*(sqrt(M_PI)*(((a - r)*(a - r)) + 2.0*alpha*t)/(2.0*alpha*t) -
2.0*(a - r)*(0.5*exp(-((a - r)*(a - r))/(4.0*alpha*t)) + (a -
r)/sqrt(alpha*t)*((a - r)*(a - r) + 2.0*alpha*t)*Gmma(gm[0],gm[1],((a - r)*(a
- r))/(4.0*alpha*t))/(4.0*((a - r)*(a -
r))))/sqrt(alpha*t))/(2.0*sqrt(M_PI));
Interm[4] = a*t*(sqrt(M_PI)*(((a + r)*(a + r)) + 2.0*alpha*t)/(2.0*alpha*t) -
2.0*(a + r)*(0.5*exp(-((a + r)*(a + r))/(4.0*alpha*t)) + (a +
r)/sqrt(alpha*t)*(((a + r)*(a + r)) + 2.0*alpha*t)*Gmma(gm[0],gm[1],((a +
r)*(a + r))/(4.0*alpha*t))/(4.0*((a + r)*(a +
r))))/sqrt(alpha*t))/(2.0*sqrt(M_PI));
Intheta = alpha*A0*(Interm[0] - Interm[1] + Interm[2] - Interm[3] +
Interm[4])/cond;

return Intheta/r;
}

double granule::OrigTemp(double t){
double Origfnc[3], gm[2], lg[2];
gm[0] = 0.5;
gm[1] = 0.0;
lg[0] = 1.5;
lg[1] = -0.5;
Origfnc[0] = t;
Origfnc[1] = t*(sqrt(M_PI)*(a*a + 2.0*alpha*t)/(2.0*alpha*t) -
2.0*a*(0.5*exp(-a*a/(4.0*alpha*t)) + sqrt(a*a/(alpha*t))*(a*a +
2.0*alpha*t)*Gmma(gm[0],gm[1],a*a/(4.0*alpha*t))/(4.0*a*a))/(sqrt(alpha)*sqrt
(t)))/sqrt(M_PI);
Origfnc[2] = sqrt(t)*(-(a*sqrt(M_PI)/(sqrt(alpha)*sqrt(t))) -
sqrt(a*a/(alpha*t))*Gmma(-gm[0],gm[1],a*a/(4.0*alpha*t))/2.0)/sqrt(M_PI);
return alpha*A0*(Origfnc[0] - Origfnc[1] - (a*Origfnc[2])/sqrt(alpha))/cond;
}

double granule::Temp(double r, double t){
if(t == 0.0)
      return 0.0;
```

```
else{
      if(r == 0.0)
            return OrigTemp(t);
      else{
            if((r>0.0)&&(r<a))
                  return InTemp(r,t);
            else{
                  if(r == a)
                        return SurfTemp(t);
                  else{
                        if(r>a)
                              return OutTemp(r,t);
                  }
            }
      }
}
}

double granule::MelTemp(double r, double t){
      double times;
      double ReturnTemp;
      for(int i =0; i<PulseVec.size(); i++){
            times = t - PulseVec[i];
/*
This part here is basically the original MelTemp Function from FORTRAN, here
it is looped over the number of pulses that are given
*/
if( times < 0.0){
      ReturnTemp += 0.0;
}
else{
      if(times > pulsedur)
                  ReturnTemp += Temp(r,times) - Temp(r,times-pulsedur);
      else
                  ReturnTemp += Temp(r,times);
}
}
return ReturnTemp;
}

BigTherm::BigTherm(char* configfilename){
boost::property_tree::ptree configfile;
boost::property_tree::ini_parser::read_ini(configfilename, configfile);
BoxLength=configfile.get<double>("Bigtherm.BoxLength");
BoxDepth=configfile.get<double>("Bigtherm.BoxDepth");
granny.a=configfile.get<double>("Bigtherm.GranularRadius");
xnum=configfile.get<int>("Bigtherm.xnum");
xmin=configfile.get<double>("Bigtherm.xmin");
xmax=configfile.get<double>("Bigtherm.xmax");
ynum=configfile.get<int>("Bigtherm.ynum");
ymin=configfile.get<double>("Bigtherm.ymin");
ymax=configfile.get<double>("Bigtherm.ymax");
znum=configfile.get<int>("Bigtherm.znum");
zmin=configfile.get<double>("Bigtherm.zmin");
zmax=configfile.get<double>("Bigtherm.zmax");
tnum=configfile.get<int>("Bigtherm.tnum");
tmin=configfile.get<double>("Bigtherm.tmin");
```

```
tmax=configfile.get<double>("Bigtherm.tmax");
MelDens=configfile.get<double>("Bigtherm.MelDens");
rnum=configfile.get<int>("Bigtherm.rnum");
absorb=configfile.get<double>("Bigtherm.absorb");
irr=configfile.get<double>("Bigtherm.irr");
granny.pulsedur=configfile.get<double>("Bigtherm.pulsedur");
granny.alpha=configfile.get<double>("Bigtherm.alpha");
granny.cond=configfile.get<double>("Bigtherm.cond");
corspot=configfile.get<double>("Bigtherm.corspot");
spotsize=configfile.get<double>("Bigtherm.spotsize");
trans=configfile.get<double>("Bigtherm.trans");
improf=configfile.get<int>("Bigtherm.improf");
SeedFlag=configfile.get<int>("Bigtherm.SeedFlag");
PulseNum=configfile.get<int>("Bigtherm.PulseNum");
PulseSep=configfile.get<double>("Bigtherm.PulseSep");
PulseVec.push_back(tmin);
for(int i =1; i<PulseNum; i++){
      PulseVec.push_back(PulseVec[i-1] + granny.pulsedur + PulseSep);
}
granny.PulseVec = PulseVec;
//-------------------------------------------------------------
TempOutputFile=configfile.get<string>("Bigtherm.TempOutputFile");
DamageOutputFile=configfile.get<string>("Bigtherm.DamageOutputFile");
DamageInfo=configfile.get<int>("Bigtherm.DamageInfo");
numthreads=configfile.get<int>("BigTherm.numthreads",1);
if(numthreads<1)
      numthreads = 1;

////////////////////////////////////////////////////////////
//end loading
////////////////////////////////////////////////
xsize = (xmax - xmin)/xnum;
ysize = (ymax - ymin)/ynum;
zsize = (zmax - zmin)/znum;
tsize = (tmax - tmin)/tnum;
melnum = 0;
melnum = configfile.get<int>("Bigtherm.Melnum", 0);
if(melnum == 0){
      MelDens=configfile.get<double>("Bigtherm.MelDens");
      melnum = MelDens * BoxLength*BoxLength * BoxDepth;
}
else
      MelDens = 0;
/*
!*********************************************************************
!
!     Determine the image profile on the retina.
!     Use image profile technique determined by
!     the value of improf:
!
!                  improf=0  <->  Gaussian Image
!                  improf=3  <->  Top Hat Source
!                  improf=4  <->  Annular Beam
!
!     The image diameter is then determined by the value of
!     spotsize.  The amount of focusing
!     that has taken place is a function of the retinal image diameter
```

```
!     as well. For annular beams, which assume a 37 per cent central
!     obscuration, the diameter of the central obscuration, dobs, is
!     spotsize*DSQRT(0.37).
!
!*********************************************************************
*/

dobs = spotsize * sqrt(0.37);
      if(improf ==  0)
            Focus = 2 * (corspot*corspot) / (spotsize*spotsize);
      if (improf == 3)
            Focus = (corspot*corspot) / (spotsize*spotsize);
      if(improf == 4)
                  Focus = (corspot*corspot) / (spotsize*spotsize);
                  granny.A0 = 3 * irr * trans * Focus * (1 - (1 - exp(-2 *
granny.a * absorb)*    (1 + 2 * granny.a * absorb))/(2 * granny.a*granny.a *
absorb*absorb) )/ (4 * granny.a * granny.pulsedur);
            sigma = 8.0 / (spotsize*spotsize);

      if (tmax < granny.pulsedur)
          Qtot = granny.A0 * 4.0 * M_PI * granny.a*granny.a*granny.a * tmax /
3.0;
      else
          Qtot  =  granny.A0  *  4.0  *  M_PI  *  granny.a*granny.a*granny.a  *
granny.pulsedur / 3.0;

vector<double> dummy;
// computing weighting of the fluence for the placed granule
if(SeedFlag == 0){
      MelPlacementFile=configfile.get<std::string>("Bigtherm.MelPlacementFile
");
      fstream posfile(MelPlacementFile.c_str());
      if(posfile.good()){
            double in;
            string throwaway;
            while(!posfile.eof()){
                  posfile>>in;
                  dummy.push_back(in);
                  posfile>>in;
                  dummy.push_back(in);
                        posfile>>in;
                  dummy.push_back(in);
                  Pos.push_back(dummy);
                  dummy.clear();
                  getline(posfile,throwaway);
            }
      }
      else{
            cout<<"Placement File not found, using random seed"<<endl;
            SeedFlag = 1;
      }
}
double thermcon;
if(SeedFlag == 1){

      srand(time(NULL));
      for( int j = 0; j<melnum; j++) {
```

```
            if(j== 0)
                  Pos.clear();
            ran1 = (rand() % 101)/100.0;
            ran2 = (rand() % 101)/100.0;
            ran3 = (rand() % 101)/100.0;
            thermcon = (ran1 * BoxLength) - (BoxLength/2.0);
            dummy.push_back(thermcon);
            thermcon = (ran2 * BoxLength) - (BoxLength/2.0);
            dummy.push_back(thermcon);
            thermcon = (ran3 * BoxDepth) - (BoxDepth/2.0);
            dummy.push_back(thermcon);
            Pos.push_back(dummy);
            dummy.clear();
      }
}

for( int j = 0; j<melnum; j++) {

      r = sqrt(Pos[j][0]*Pos[j][0] + Pos[j][1]*Pos[j][1]);

      if (improf == 0)
               weight.push_back(exp(-sigma * r*r));
      else if (improf == 3){
            if (r <=  spotsize/2.0){
                  weight.push_back(1.0);
            }
            else{
                  weight.push_back(0.0);
            }
      }
      else if (improf == 4){
            if( r > spotsize/2.0)
                  weight.push_back(0.0);
            else {
                  if(r < dobs/2.0)
                        weight.push_back(0.0);
                  else
                        weight.push_back(1.0);
            }
      }
}
for(int i = 0; i<=tnum; i++){

      t = tmin + i * tsize;
      rsize.push_back(0.0);

        if (t <= 0.00001)
            rsize[i] = (0.0005/rnum);
         else
            rsize[i] = (0.0005*(7.0 + log10(t))/rnum);

      for(int j = 0; j<=rnum; j++){
            r = j * rsize[i];
                  dummy.push_back(granny.MelTemp(r,t));
                  if(dummy[j] + Tbody > 100)
                  TempFlag = 1;
      }
```

```
      VecTemp.push_back(dummy);
      dummy.clear();
}
// ---------  Check conservation of energy --------------
if (tmax <= 10.0E-6)
      dx = 0.0005/rnum;
else
      dx = 0.0005*(7.0 + log10(tmax))/rnum;
         Qkept = 0.0;

for(int j =0; j<rnum; j++){
      Qkept = Qkept + VecTemp[tnum][j] * 4.1868 * 4.0 * M_PI * dx*dx*dx *
(3.0*(j+1)*(j+1) - 3.0*(j+1) + 1.0) / 3.0;
}
Qlost = Qtot - Qkept;

      if (TempFlag == 1) {
            cout<<endl;
            cout<<"////////////////////////////////////////////////"<<endl;
            cout<<"--------------------------------"<<endl;
            cout<<"      Warning . . . 100 degrees exceeded."<<endl;
            cout<<"--------------------------------"<<endl;
      cout<<"////////////////////////////////////////////////"<<endl;
      }
}

double BigTherm::BigThermTempAt(double x, double y, double z, double t){
//WARNING this function will NOT give outputs for arbitrary t, simply the
//closest t to the tsize given away from the min t.
if((t<tmin)||(t>tmax))
      return -1;
int l = (t-tmin)/tsize;
double dist, ilook, Temper, SumTemp;
SumTemp=0.0;

//!-------- Summing effects of all granules---------------
omp_set_num_threads(numthreads);
#pragma omp parallel for reduction(+:SumTemp) private(dist, ilook, Temper)
for(int m = 0; m<melnum; m++){
      dist = sqrt((Pos[m][0]-x)*(Pos[m][0]-x) + (Pos[m][1]-y)*(Pos[m][1]-y) +
(Pos[m][2]-z)*(Pos[m][2]-z));

//!--------- Individual Temps Interpolated From VecTemp Table----
      ilook = dist/rsize[l];
      if (ilook > rnum-1) {
            Temper=0.0;
      }
      else{
            Temper  =  VecTemp[l][ilook]  +  (dist  -  ilook  *  rsize[l])  *
(VecTemp[l][ilook+1] - VecTemp[l][ilook]) / rsize[l];
      }

      SumTemp += weight[m] * Temper;

}
/*
```

```
Slight issues with noise, method only accurate to about a hundredth of a
degree anyway so throwing out temperature rises on this magnitude cleans up
the
profiles a bit.
*/
if(SumTemp < 1e-6)
      return 0.0;

return SumTemp;
}

//This function is a copy of the constructor if you want to run it again with
the //different beam parameters

void BigTherm::ReConstruct(void){
      double thermcon;
      vector<double> dummy;
      Pos.clear();
      srand(time(NULL));
      for( int j = 0; j<melnum; j++) {
            if(j== 0)
                  Pos.clear();
            ran1 = (rand() % 101)/100.0;
            ran2 = (rand() % 101)/100.0;
            ran3 = (rand() % 101)/100.0;
            thermcon = (ran1 * BoxLength) - (BoxLength/2.0);
            dummy.push_back(thermcon);
            thermcon = (ran2 * BoxLength) - (BoxLength/2.0);
            dummy.push_back(thermcon);
            thermcon = (ran3 * BoxDepth) - (BoxDepth/2.0);
            dummy.push_back(thermcon);
            Pos.push_back(dummy);
            dummy.clear();
      }

PulseVec.clear();
PulseVec.push_back(tmin);
for(int i =1; i<PulseNum; i++){
      PulseVec.push_back(PulseVec[i-1] + granny.pulsedur + PulseSep);
}
granny.PulseVec = PulseVec;
weight.clear();
for( int j = 0; j<melnum; j++) {
      r = sqrt(Pos[j][0]*Pos[j][0] + Pos[j][1]*Pos[j][1]);
      if (improf == 0)
               weight.push_back(exp(-sigma * r*r));
      else if (improf == 3){
            if (r <=  spotsize/2.0){
                  weight.push_back(1.0);
            }
            else{
                  weight.push_back(0.0);
            }
      }
      else if (improf == 4){
            if( r > spotsize/2.0)
                  weight.push_back(0.0);
```

```
            else {
                  if(r < dobs/2.0)
                        weight.push_back(0.0);
                  else
                        weight.push_back(1.0);
            }
      }
}
VecTemp.clear();
for(int i = 0; i<=tnum; i++){
      t = tmin + i * tsize;
      rsize.push_back(0.0);
        if (t <= 0.00001)
            rsize[i] = (0.0005/rnum);
         else
            rsize[i] = (0.0005*(7.0 + log10(t))/rnum);

      for(int j = 0; j<=rnum; j++){

            r = j * rsize[i];
                  dummy.push_back(granny.MelTemp(r,t));
                  if(dummy[j] + Tbody > 100)
                  TempFlag = 1;
      }
      VecTemp.push_back(dummy);
      dummy.clear();

}
// -----------  Check Conservation of Energy -----------------
if (tmax <= 10.0E-6)
      dx = 0.0005/rnum;
else
      dx = 0.0005*(7.0 + log10(tmax))/rnum;
         Qkept = 0.0;

for(int j =0; j<rnum; j++){
      Qkept = Qkept + VecTemp[tnum][j] * 4.1868 * 4.0 * M_PI * dx*dx*dx *
(3.0*(j+1)*(j+1) - 3.0*(j+1) + 1.0) / 3.0;
}
Qlost = Qtot - Qkept;

if (TempFlag == 1) {
      cout<<endl;
      cout<<"////////////////////////////////////////////////"<<endl;
      cout<<"------------------------------------------"<<endl;
      cout<<"      Warning . . . 100 degrees exceeded."<<endl;
      cout<<"------------------------------------------"<<endl;
      cout<<"////////////////////////////////////////////////"<<endl;
}
}

void BigTherm::BigTherm2(void){
/*
This will run the equivalent of the Original bigtherm2, and write it to the
file listed for output in the config file. It writes the entirety of the box
and the entirety of the given time.
*/
```

```
ofstream writefile(TempOutputFile.c_str(),ios::out);
writefile<<"t\t"<<"x\t"<<"y\t"<<"z\t"<<"Temp"<<"\n";
for(int i = 0; i<xnum+1; i++){
      x = xmin + i * xsize;
      for(int ii = 0; ii<ynum+1; ii++){
            y = ymin + ii*ysize;
            for(int iii = 0; iii<znum+1; iii++){
                  z = zmin + iii*zsize;
                  for(int iv = 0; iv < tnum+1; iv ++){
                        t = tmin + iv * tsize;
                        double thing = BigThermTempAt(x,y,z,t);

      writefile<<t<<"\t"<<x<<"\t"<<y<<"\t"<<z<<"\t"<<thing<<endl;
                  }
            }
      }
}
}

/*
This runs BigTherm2 for a given subset of the box. This can be used so that
one does not output or calculate an unneeded data, i.e. points far out enough
from the beam that there is no significant temperature rise
*/
void BigTherm::BigTherm2(double outxmin, double outxmax, int outxnum, double
outymin, double outymax, int outynum, double outzmin, double outzmax, int
outznum){
ofstream writefile(TempOutputFile.c_str(),ios::out);
writefile<<"t\t"<<"z\t"<<"x\t"<<"y\t"<<"Temp"<<"\n";
double outysize, outxsize, outzsize;
outysize =(outymax - outymin)/outynum;
outxsize =(outxmax - outxmin)/outxnum;
outzsize =(outzmax - outzmin)/outznum;
//using dummy variables in case one wants to use bigtherm2(void) afterwards
for(int i = 0; i<tnum; i++){
        t = tmin + i * tsize;
        for(int ii = 0; ii<outznum; ii++){
                z = outzmin + ii * outzsize;
                for(int iii = 0; iii<outxnum; iii++){
                        x = outxmin + iii*outxsize;
                        for(int vi = 0; vi<outynum; vi++) {
                                y = outymin + vi*outysize;
                                double thing = BigThermTempAt(x,y,z,t);

writefile<<t<<"\t"<<z<<"\t"<<x<<"\t"<<y<<"\t"<<thing<<"\n";
                        }
                }
        writefile<<"\n";
        }
}

}
void takatadamage(char* configfilename){

      int DamFlag;
      int xnum, ynum, znum, tnum;
      double Pi, BoxLength, BoxDepth, a, MelDens;
```

```
      double xmin, xmax, xsize;
      double ymin, ymax, ysize;
      double zmin, zmax, zsize;
      double tmin, tmax, tsize;
      vector< double > Temp;
      double c11, c21, c12, c22;
      double irr,pulsedur,E;
      double x, y, z, t, Tbody, rnum, absorb;
      double TempA;
      double spotsize,spotsizex,spotsizey;
      float Term1,Term2;
      float Sum1, Sum2;
      float Integrand;
      float Damage;
      int old;
      string inputfile;
      string outputfile;
      boost::property_tree::ptree configfile;
      boost::property_tree::ini_parser::read_ini(configfilename, configfile);
      string OutputFile;

//-----initializing the output file

      outputfile = configfile.get<string>("Bigtherm.DamageOutputFile");
      fstream out(outputfile.c_str());
      out.open(outputfile.c_str());
      out<<"t\t"<<"x\t"<<"y\t"<<"z\t"<<"Damage\t"<<endl;

//!-----Initializing constants:

      Tbody=37.0;
      DamFlag=0;
      c11=149.0;
      c12=50000.0;
      c21=242.0;
      c22=80000.0;

//-----Reading input variables-------

      BoxLength=configfile.get<double>("Bigtherm.BoxLength");
      BoxDepth=configfile.get<double>("Bigtherm.BoxDepth");
      xnum=configfile.get<int>("Bigtherm.xnum");
      xmin=configfile.get<double>("Bigtherm.xmin");
      xmax=configfile.get<double>("Bigtherm.xmax");
      ynum=configfile.get<int>("Bigtherm.ynum");
      ymin=configfile.get<double>("Bigtherm.ymin");
      ymax=configfile.get<double>("Bigtherm.ymax");
      znum=configfile.get<int>("Bigtherm.znum");
      zmin=configfile.get<double>("Bigtherm.zmin");
      zmax=configfile.get<double>("Bigtherm.zmax");
      tnum=configfile.get<int>("Bigtherm.tnum");
      tmin=configfile.get<double>("Bigtherm.tmin");
      tmax=configfile.get<double>("Bigtherm.tmax");
      MelDens=configfile.get<double>("Bigtherm.MelDens");
      rnum=configfile.get<int>("Bigtherm.rnum");
      old=configfile.get<int>("Bigtherm.old");
      absorb=configfile.get<double>("Bigtherm.absorb");
```

```
//!-----Calculating necessary parameters--------

      xsize = (xmax - xmin)/xnum;
      ysize = (ymax - ymin)/ynum;
      zsize = (zmax - zmin)/znum;
      tsize = (tmax - tmin)/tnum;
/*
!**********************************************************************
!     Calculate Damage(x,y,z,t)
!**********************************************************************
*/
// initializing input file
      inputfile = configfile.get<string>("Bigtherm.tempfile");
      ifstream damagefile(inputfile.c_str());
      double dummy;
      string line;
      int dam;
if( old ==0)
      getline(damagefile, line);

//!------Stepping in x direction------------

      for(int i=0;i<xnum+1;i++){
            x = xmin + i * xsize;

//!---------Stepping in y direction------------

            for(int ii = 0; ii<ynum+1; ii++){
                  y = ymin + ii * ysize;

//!------------Stepping in z direction-------------

                  for(int iii = 0; iii<znum+1; iii++){
                        z = zmin + iii * zsize;

//!---------------Stepping in time---------------------

                        for(int iv = 0; iv<tnum+1; iv++){
                              t = tmin + iv * tsize;

                              if(old ==1){
                                    damagefile >> dummy;
                                    Temp.push_back(dummy);
                              }

                              if(old == 0){
                                    damagefile
>>dummy>>dummy>>dummy>>dummy>>dummy;
                                    Temp.push_back(dummy);
                              }
                        }
//!------------Simpson's Rule for Damage Integral---------------

Sum1=0.0;
Sum2=0.0;
```

```
if(Tbody + Temp[0] < 50.0)
      Term1=exp(c11 - c12/(273 + Tbody + Temp[0]));
else
      Term1=exp(c21 - c22/(273 + Tbody + Temp[0]));

if(Tbody + Temp[tnum-1] < 50.0)
      Term2=exp(c11 - c12/(273 + Tbody + Temp[tnum-1]));
else
      Term2=exp(c21 - c22/(273 + Tbody + Temp[tnum-1]));

for(int iv =1; iv<tnum; iv+=2){
      if ((Tbody + Temp[iv]) < 50.0)
            Integrand=exp(c11 - (c12/(273 + Tbody + Temp[iv])));
      else
            Integrand=exp(c21 - c22/(273 + Tbody + Temp[iv]));
            Sum1=Sum1+Integrand;
}

Sum1*=4.0;

for(int iv =2; iv<tnum-1; iv +=2){
      if ((Tbody + Temp[0]) < 50.0)
            Integrand = exp(c11 - (c12/(273 + Tbody + Temp[iv])));
      else
            Integrand=exp(c21 - c22/(273 + Tbody + Temp[iv]));
            Sum2+=Integrand;
}

Sum2*=2.0;

                        Damage=tsize*(Term1 + Term2 + Sum1 + Sum2)/3.0;

                        if(Damage >= 1.0)
                              dam = 1;
                        else
                              dam = 0;

      cout<<x<<"\t"<<y<<"\t"<<z<<"\t"<<Damage<<endl;

                        if (Damage >=1.0)
                              DamFlag=1;
                        Temp.clear();
            }
      }
}
      out.close();
      damagefile.close();
}
void BigTherm::TakataDamage(void){

      int DamFlag;
      vector< double > Temp;
      double c11, c21, c12, c22;
      double x, y, z, t;
      double TempA;
      double spotsize,spotsizex,spotsizey;
      float Term1,Term2;
```

```
      float Sum1, Sum2;
      float Integrand;
      float Damage;
      int old;

//-----initializing the output file

      ofstream DamageOut(DamageOutputFile.c_str(),ios::out);
      ofstream TempOut(TempOutputFile.c_str(),ios::out);

      DamageOut << "x\t" << "y\t" << "z\t" <<"Damage\t" <<endl;
      if(DamageInfo == 1){
            TempOut<<"t\t"<<"x\t"<<"y\t"<<"z\t"<<"Temp\t"<<endl;
      }

//!-----Initializing constants:

      Tbody=37.0;
      DamFlag=0;
      c11=149.0;
      c12=50000.0;
      c21=242.0;
      c22=80000.0;

//!-----Calculating necessary parameters--------

      xsize = (xmax - xmin)/xnum;
      ysize = (ymax - ymin)/ynum;
      zsize = (zmax - zmin)/znum;
      tsize = (tmax - tmin)/tnum;
/*
!**********************************************************************
!     Calculate Damage(x,y,z,t)
!**********************************************************************
*/

// initializing input file
      double dummy;
      int dam;

//!------Stepping in x direction------------

      for(int i=0;i<xnum+1;i++){
            x = xmin + i * xsize;

//!---------Stepping in y direction------------

            for(int ii = 0; ii<ynum+1; ii++){
                  y = ymin + ii * ysize;

//!------------Stepping in z direction-------------

                  for(int iii = 0; iii<znum+1; iii++){
                        z = zmin + iii * zsize;

//!---------------Stepping in time---------------------
```

```
                        for(int iv = 0; iv<tnum+1; iv++){
                              t = tmin + iv * tsize;
                              dummy = BigThermTempAt(x,y,z,t);
                              Temp.push_back(dummy);
                              if(DamageInfo == 1)

TempOut<<t<<"\t"<<z<<"\t"<<x<<"\t"<<y<<"\t"<<dummy<<endl;
                        }
//!------------Simpson's Rule for Damage Integral---------------

      Sum1=0.0;
      Sum2=0.0;

      if(Tbody + Temp[0] < 50.0)
            Term1=exp(c11 - c12/(273 + Tbody + Temp[0]));
      else
            Term1=exp(c21 - c22/(273 + Tbody + Temp[0]));

      if(Tbody + Temp[tnum-1] < 50.0)
            Term2=exp(c11 - c12/(273 + Tbody + Temp[tnum-1]));
      else
            Term2=exp(c21 - c22/(273 + Tbody + Temp[tnum-1]));

for(int iv =1; iv<tnum; iv+=2){

      if ((Tbody + Temp[iv]) < 50.0)
            Integrand=exp(c11 - (c12/(273 + Tbody + Temp[iv])));
      else
            Integrand=exp(c21 - c22/(273 + Tbody + Temp[iv]));
      Sum1 = Sum1+ Integrand;
}
      Sum1*=4.0;
for(int iv =2; iv<tnum-1; iv +=2){
      if ((Tbody + Temp[0]) < 50.0)
            Integrand = exp(c11 - (c12/(273 + Tbody + Temp[iv])));
      else
            Integrand = exp(c21 - c22/(273 + Tbody + Temp[iv]));
      Sum2+=Integrand;
}

      Sum2*=2.0;
      Damage=tsize*(Term1 + Term2 + Sum1 + Sum2)/3.0;
      if(Damage >= 1.0)
            dam = 1;
      else
            dam = 0;
      DamageOut<<x<<"\t"<<y<<"\t"<<z<<"\t"<<dam<<endl;
      if (Damage >=1.0)
            DamFlag=1;
      Temp.clear();
            }
      }
}
      DamageOut.close();
      TempOut.close();

}
```

7.4 Input File

```
;     Text preceeded by or ";" are comments and will not be read by
;     boost_property_tree parser.  This example will generate a temperature
;     profile and damage spot from a given seed.

[Bigtherm]
BoxLength= 0.0040
BoxDepth = 0.0014
GranularRadius= 0.0001
xnum=40
xmin=-0.0020
xmax=0.0020
ynum=40
ymin=-0.0020
ymax=0.0020
znum=2
zmin=-0.0007
zmax=0.0007
tnum=20
tmin=0.0
tmax=0.002
Melnum=
MelDens=8.33e9
rnum=500
absorb=4000.0

;0.0040     0.0014      .0001  # BoxLength, BoxDepth, Granular Radius(a)
;40   -.0020      .0020  # Number of x steps(xnum), xmin, xmax
;40   -.0020      .0020  # Number of y steps(ynum), ymin, ymax
;2    -.0007  .0007  # Number of z steps(znum), zmin, zmax
;20   0.0   .002    # Number of time steps(tnum), tmin, tmax
;8.33D9     500     4000.0  # MelDens, rnum, absorb


;  Comments here will not be read by C++
;
;   1) Input all distances in cm and all times in sec.
;      Melanin density(MelDens) in granules/cm**3.
;      Melanin absorption coefficient(absorb) in cm**-1.
;      All calculations are done in cgs units. Some energies are input in
;      Joules; but these are converted to calories before calculation.
;
;   2) Melanin Parameters
;       A. Melanosome (or granular) radius is roughly 1 micron. This is
;          a variable input parameter.
;       B. Melanosomes are distributed through a rectangular volume whose
;          x,y dimensions are BoxLength and z dimension is BoxDepth.
;          Granules are randomly distributed through this space using a
;          Cartesian coordinate system with origin at the geometric center
;          of the volume. Granular volumes do not overlap.
;       C. The volume of an RPE cell is independent of the melanin dist.
;          volume. It is estimated to be 20 x 20 microns cross section(x,y)
;          and 15 microns depth(z) = 6 x 10**-9 cm**3. The portion of the
;          RPE cell volume that contains melanosomes is Vm = 20 microns x
;          20 microns x BoxDepth.
```

```
;         D. The melanin number density, Nm, is the number of granules/RPE
cell.
;             The melanin volume density, MelDens, may be computed from Nm
using:
;            MelDens(granules/cm**3) = Nm(granules/cell)/Vm(cm**3/cell).
;             For example, BoxDepth = 15 microns and Nm = 120 granules/cell
gives
;          Vm = 6E-9 cm**3/cell and MelDens = 2E10 granules/cm**3.
;       E. The number of granules in the whole dist. vol.(melnum) is:
;            melnum = MelDens * BoxLength**2 * BoxDepth
;          To avoid excessive cpu time, this number should not exceed 20,000.
;          (The code is currently hard-wired for arrays of dimension 20,000).
;
;   3) The single granule temperature distribution, T(r,t) is calculated
;      using a spherical coordinate system independent of the Cartesian
;      system used for melanin dist. The calculation is radially symmetric
;      and is done for rnum radial steps and tnum time steps. Note that
;      T(r,t) is analytically calculated and is not the product of a temporal
;      summation or a spatial stepping algorithm. Each value can thus be
;      calculated independently of values at other positions and times.
;
;   4) The final temperature distribution in the retina is calculated using
;      the same Cartesian coordinate system as for melanin dist. The value
;      T(x,y,z,t) at a certain arbitrary position and time is the linear
;      superposition of the contributions of all the granules T(r1,t)+T(r2,t)
;          +T(r3,t)+... at that time, where r1,r2,r3,etc. are the radial
distances
;      from each granule to (x,y,z). The value T(x,y,z,t), like T(r,t), can
;      be calculated independently for each position and time.
;
;   5) Since each T(x,y,z,t) is independently calculated, the space and time
;      grid chosen for the temperature calculation is in theory arbitrary.
;              Note that the number of steps in each dimension is
xnum,ynum,znum,tnum;
;      whereas the actual number of grid points is xnum+1,ynum+1,znum+1, and
;      tnum+1. The do loops in the code go from 0 to xnum,ynum,etc. and not
;      from 1 to xnum+1,etc.
;   6) The number of x,y,& z grid points is only constrained by the spatial
;       resolution needed for visualization. Max and min values may be inside
;      or outside the melanin volume. They need not be symmetric about the
;      origin. They are not tied to the maximum value of r used in
calculating ;      T(r,t) for one granule. For r>rmax the contribution of
that granule is ;  simply 0.
;
;   7) Typically xnum & ynum should be set to give at least 1 step/micron. A
;       minimum of znum+1 = 3 should be used where the 3 z slices represent
the
;       top, middle, and bottom layers of the RPE. More z steps may be needed
;      to accurately construct a side view of the damage volume or to
;      accurately account for true shadowing.
;
;   8) The number of time steps tnum is constrained by the requirements of
;        the damage calculation. Temporal resolution must be great enough to
;      give convergance of the Arrenhius integral over the time-temperature
;      history.  Typically tnum of about 20-40 is sufficient. Unlike the
;      spatial grid the temporal calculation is limited to those time values
;      for which T(r,t) has previously been calculated.
```

```
;
;   9) The minmum, tmin, is always 0 (before the pulse). The max value can be
;         inside or outside the pulse interaction time, if the only thing
desired
;          is the temperature distribution. For damage calculations to be
accurate
;        however, tmax must be large enough that the tissue has cooled to
normal
;          (37 C). For most cases the pulsewidth plus 2 msec should be
sufficient.
;      Very long pulses (on the order of seconds) or very large spots may
;      reqiuire more than 2 msec cooling time and adjustments may be needed.
;      If tmax is large, the energy conservation report will likely show
;      that much energy has been lost.  This simply means that energy has
;      had time to move out of the RPE and into the surrounding tissue.
;
;   10) It is not a good idea to run this model for times, t < 1 microsec.
;         (The pulse duration can be as short as you desire, but in the
;       ultrashort case, don't ask for a temperature profile until well after
;       the pulse has expired.  We do not expect that thermal mechanisms are
;       relevant for ultrashort anyway.)
;
;   11)The Arrhenius Integral damage calculations are done using a Simpson's
;      Rule Integrater. The coefficients are from either Takata or Welch.
;
;   12)Image diameter- The BoxLength should be greater than or equal to the
;      transverse beam diameter of the laser; i.e., the beam should encounter
;        melanin everywhere it falls on the retina. It is probably best if
twice
;      the laser diameter is used, so that all the beam is covered out to low
;      intensities; however, this not strictly necessary for validity.
;
;   13)Run Time - There are basically four parts to the thermal code:
;
;        i. Random granule placement - varies with melnum.
;       ii. Depth averaging or shadowing - varies with melnum.
;      iii. T(r,t) single granule calc. - varies with tnum.
;       iv. T(x,y,z,t) mult. gran. calc.- varies with xnum, ynum, znum, tnum,
;           melnum
;
;      Parts i,ii, and iii do not take a high percentage of cpu time even at
;      fairly high values of melnum and tnum. Almost all cpu time is spent on
;      T(x,y,z,t) which is linear in xnum,ynum,znum,tnum, and melnum.


;information formerly contained in the inp2 files
irr=3.0e-5
;(irr)Corneal Fluence in J/cm**2.
pulsedur=3.0e-6
; (pulsedur)Pulse Duration in sec
alpha=0.00132798
; (alpha)Thermal Diffusivity of Medium in cm**2/sec.
cond=0.00556
; (cond)Thermal Conductivity of Medium in J/cm-C-sec.
corspot=0.7
; (corspot)Beam Diameter at cornea in cm.
spotsize=0.0022
```

```
; (spotsize)Retinal Image Diameter at 1/e**2 points in cm.
trans=1.0
; (trans)Fraction of pulse energy transmitted to retina.
improf=0
; (improf)Image Type[0=Gaussian, 3=Top Hat, 4=Annular].
SeedFlag=1
; SeedFlag [0=Fixed Seed, 1 = Random Seed from time() function].

;   1) Input all distances in cm and all times in sec.
;      All calculations are done in cgs units. Some energies are input in
;      Joules; but these are converted to calories before calculation.
;
;   2) Corneal beam diameter is used to compute the Total Interocular
;      Energy (multiplying area by Corneal Fluence).  Therefore, if
;      the beam is larger than the pupil, the pupil diameter should be used.
;
;   3) The average focusing power of the eye can be determined by:
;      Retinal Fluence / Corneal Fluence = (Corneal Diam / Retinal Diam)**2.
;        For 10**5, you might choose Corneal Diam = .7 and Retinal Diam =
.0022.
;
;   4) Fraction of pulse energy transmitted is a function of wavelength and
;      may be obtained from the literature.  (See Maher)
;
;   6) Values of thermal diffusivity and conductivity for water:
;            0.00132798 cm**2/sec and 0.00556 J/cm-C-sec
;file for explicit placement of melinen granules
MelPlacementFile=ExampleMelPlacementFile.txt

TempOutputFile=ExampleOutputTempFile.temp
;name for output temperature file, the file extention is arbitrary
;output will be formated time, x, y, z, temperature. units are the same as
above
DamageOutputFile=ExampleOutputDamageFile.dam
;name for output damage file, the file extention is arbitrary,
;output will be x, y, z then either a 0 or 1
;0 indicates no thermal damage has been done, 1 indicates damage has been
done
PulseNum=1
;The number of pulses long an exposure is, defauts to 1 if none is given
PulseSep=0.001
;the time in seconds between each pulse.
tempfile=ExampleInputTempFile.temp
;If you're using the simple re-implementation of the takata model, this is
the temperature file you need for that.
old=0
;this was an option that's only needed in the re-implementation of the takata
model, then you need to set this flag to 1 in order to use the old fortran
temp files.
DamageInfo=1
;set this flag to 1 if you want the temperature file generated for the damage
to be output as well
```